SLAC-151
UC-34
(EXP) (EXPI)

ON THE USE OF A MACRO PROCESSOR WITH SUMX

JOSEPH C. H. PARK

STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERSITY
Stanford, California 94305

and

MAX-PLANCK-INSTITUT FÜR PHYSIK UND ASTROPHYSIK
Munich, Germany

PREPARED FOR THE U. S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT NO. AT(04-3)-515

June 1972

Printed in the United States of America. Available from National Technical Information Service, U. S. Department of Commerce, 5295 Port Royal Road, Springfield, Virginia 22151.
Price: Printed Copy $3.00; Microfiche $0.95.

ACKNOWLEDGEMENTS

I am grateful to Professor R. F. Mozley of the Stanford Linear Accelerator Center for encouragement in writing this report and to Professor H. Billing of the Max-Planck-Institut für Physik und Astrophysik, where part of the work was done. It was a pleasure to work with Mr. J. Ahern as members of the experimental group D at SLAC.

TABLE OF CONTENTS

		Page
I.	Introduction	1
II.	An Example	3
III.	Limitations and Errors	11
	Appendix A	14
	Appendix B	21
	Appendix C	24
	References	26

LIST OF TABLES

		Page
I.	Input Text	4
II.	Output Text	5
III.	Macro Library	7

I. INTRODUCTION

If the SUMX[1] control statements are considered as forming a base "language", then the power of equipping it with a macro facility becomes obvious. In the following we describe a simple scheme which consists of making one preprocessing pass through a general purpose (that is, base-language-independent) macro processor called MACROS.[2] The input text is prepared using base language statements (preferably in terms of variable symbols rather than fixed values) interspersed with preprocessor statements, such as macro definitions, macro calls, and value assignments to symbols. Prior to processing, this text is "compiled" by MACROS into a target text consisting entirely of base language statements.

For readers who are not familiar with, for example, macro-assemblers we list some of the advantages thus gained:

(1) Program parameterization.

This is achieved explicitly through the use of symbols for such quantities as number of channels, channel width, etc., the value assignments for which are delayed until the run time so that it is trivial to change them to any desired values. Furthermore names (symbols) are more convenient than numbers (BOUT locations).

(2) Shorthand notation and repetitive text generation.

This aspect of a macro processor is like SUBROUTINE in FORTRAN, apart from in- vs. out-of-line distinction. Many base language statements can be compressed into one macro call, if necessary, with variable arguments.

(3) Library facility.

Definitions for general purpose or frequently occurring macros may be collected into an external library to be shared among different jobs.

As discussed in Ref. 3 the advantages listed above are common features of macro processors. There are further advantages peculiar to our application as described below.

For experiments with large statistics it is desirable to minimize the "length" of the input data set (Data Summary Tape). One solution is to have for each event on a DST only the essential physics information such as the energy-momentum four-vectors and not the derivable quantities like invariant masses, momentum-transfers, and decay angles. They are then calculated by means of CHARMs during the SUMX run. A peculiar advantage arises in our scheme because one and the same macro call is used to generate names for quantities of interest, prepare the corresponding CHARM calls to calculate them, and assign BOUT locations for storing the named results. The chance for error is thus greatly reduced and since each run is explicitly self-contained it is easy to cross-check.

The present scheme was developed by John Ahern and the author as members of experimental group D at the Stanford Linear Accelerator Center and has been in use at SLAC since 1968.

As described elsewhere[2] MACROS is written in PL/I taking advantage of its list-processing facility with a small part dealing with the operating system such as that for requesting core-space in assembler language. All CHARMs and SUMX related routines are written in FORTRAN IV except those "pots and pans" ones such as vector- and matrix-manipulating routines which are coded in assembler language. These programs are running on the IBM system 360/91 as implemented at SLAC and with some minor differences on a similar system at Max Planck Institut fur Physik und Astrophysik.

The capability of the present version of the macro processor, MACRO01, is somewhat limited because it lacks macro-time features such as macro-time variables (to do arithmetic with) and conditional branches (to be able to define macros recursively).

In Section II our scheme is described by an actual example with enough variety to illustrate most features of MACROS. Appendix A gives a formal description of the processor for reference purposes. In Appendix B we briefly summarize various CHARMs used in the example. In Section III several examples of errors in using the processor are collected, which also serve to show the limited capability of MACRO01. Appendix C has models for Job Control Language statements required to run.

II. AN EXAMPLE

We illustrate the use of MACROS in SUMX by the example of Table I (referred to as A), which consists mostly of preprocessor statements (PPS) mixed with SUMX control (base language) statements. After one pass through MACROS this text turns into the target text consisting entirely of the base language statements as shown in Table II (referred to as B), which is then used to control SUMX. The macro library used in this example is shown in Table III (referred to as L).

Each PPS is one card long (columns 2-80) and is identified by the warning marker % in the first column. The use of the warning marker serves to speed up the processing time. Comments can be added after the marker ; as in line 1A. Line 2A calls for a macro TWINKLE. Since it is not defined in the text scanned so far the macro library is searched. MACROS being a one-pass processor macros must be defined and values must be assigned to symbols before they are used. This has the advantage of allowing local redefinitions and reassignments.

Lines 96-102L for TWINKLE show how to define a macro. Symbols (escape names) to be substituted are preceded by the escape character &, double escape characters && meaning the value must be placed starting at the specified column (left-justified). In MACROS no distinction is made between global (inter-macro) and local (within macro) symbols; any of &A, &B, or &C in TWINKLE may be left out of the parameter list, in which case values can be assigned by the assignment (=) statement appearing anywhere before use.

A macro definition is terminated by MEND, except when it is followed by another macro definition. In such case the missing MEND is assumed by the processor, since the present version does not allow a nested macro definition.

Returning to TWINKLE the parameter A can be a character string like '1000, 100' or 'KEEP 2'. Unsigned integers and fixed point numbers (digits with a decimal point) do not need be surrounded by ' as in the second argument in TWINKLE. Line 1A expands into lines 1-6B.

Line 3A calls for a macro LITTLE which, as defined in lines 103-108L, illustrates nested macro calls. Other macro calls up to line 7A are similarly straightforward. As seen in the expansions (lines 7-62B) the main purpose

TABLE I

INPUT TEXT

```
%; BEGIN 3 PRONG SUMX DECK
% CALL TWINKLE('1000,200',1,'PHOTON+P ... P + 2 PIONS 4/72')
% CALL LITTLE(.005)
% CALL     T33
% CALL     MT3('AND',1,14,12,4,'PION CHANELS')
% CALL STAR
% CALL      MT4('AND',1,14,12,4,29,'E>5.5')
% CALL C33
*SELECT
% CALL BET(40,M45,.62,.86,'RHO')
*BLOCK6
% CALL SYMBOL
% WGT=10
%          MACRO ANGLE(C,CS,PH)
%          CALL COS(C,CS)
%          CALL PHI(C,PH)
%          MACRO WONDER(N)
EVA         &&N
%          CALL MASS('P PI-',1.08,M34)
%          CALL MASS('P PI+',1.08,M35)
%          CALL MASS('PI PI',.28,M45)
% NPT=40
%           CALL DELSQ('RHO',D45)
%          CALL ANGLE('RHO IN T-CHAN HEL FRAME',CJ45,PJ45)
%          CALL ANGLE('RHO IN S-CHAN HEL FRAME',CH45,PH45)
% NPT=''
FINISH      1
%          MEND
% CALL WONDER(26)
% CALL WONDER(25)
% CALL WONDER(24)
% CALL WONDER(23)
*ALL DONE
```

TABLE II

OUTPUT TEXT

```
*NEW PASS 1000,200
 PHOTON+P ... P + 2 PIONS 4/72
*DISCARD  1
*TAPE
 10
*SELECT
TEST      14                                       WEIGHT
          10        BIG       0
TEST      12                                       PROB
          7         BIG       .005
TEST      1
          14        TRUE
AND       12        TRUE
TEST      16                                       RJCT BY ION
          -14       EQU       -1
TEST      18                                       PROTON IDENT BY ION
          -14       EQU       1
TEST      2                                        P PI+ PI-UNIQUE
          -4        EQU       301
          -4        EQU       302
TEST      3                                        P PI+ PI-AMB
          -4        EQU       -301
          -4        EQU       -302
TEST      4                                        P PI+ PI-ALL
          2         TRUE
          3         TRUE
TEST      6                                        P K+ K-UNIQUE
          -4        EQU       303
          -4        EQU       304
TEST      7                                        P K+ K-AMB
          -4        EQU       -303
          -4        EQU       -304
TEST      8                                        P K+ K-ALL
          6         TRUE
          7         TRUE
TEST      9                                        P PBAR P
          -4        EQU       305
          -4        EQU       -305
TEST      1                                        PION CHANELS
          14        TRUE
AND       12        TRUE
AND       4         TRUE
*CHARM
SETUP               8         701
*SELECT
TEST      22
          701       BET       4.5       5.5
TEST      23
          701       BET       5.5       7.
TEST      24
          701       BET       7.        9.
TEST      25
          701       BET       9.        12.
TEST      26
          701       BET       12.       30.
TEST      29
          701       BET       5.5       30.
TEST      1                                        E>5.5
          14        TRUE
AND       12        TRUE
AND       4         TRUE
AND       29        TRUE
*CHARM
M34                 2         721       2          34                  701
M35                 2         722       2          35                  701
```

Table II (continued)

```
 66  M45                  2         723       2         45                  701
 67  COSP-       0        3         724       1         5          1        701
 68  COSP+       0        3         727       1         4          1        701
 69  COS+-       0        3         730       1         3          2        701
 70  A45                  4         754       2         45         12       701
 71  A34                  4         734       2         34         21       701
 72  A35                  4         744       2         35         21       701
 73  *SELECT
 74  TEST        40                                     RHO
 75              723      BET       .62       .86
 76  *BLOCK6
 77  EVA         26
 78     INVARIANT MASS OF P PI-
 79  100         .04      1.08
 80  721         10
 81     INVARIANT MASS OF P PI+
 82  100         .04      1.08
 83  722         10
 84     INVARIANT MASS OF PI PI
 85  100         .04      .28
 86  723         10
 87    DELSQ OF RHO
 88  100         .02                40
 89  731         10
 90     COSINE OF RHO IN T-CHAN HEL FRAME
 91  40          .05      -1.       40
 92  756         10
 93     PHI OF RHO IN T-CHAN HEL FRAME
 94  24          15.      -180.     40
 95  757         10
 96     COSINE OF RHO IN S-CHAN HEL FRAME
 97  40          .05      -1.       40
 98  758         10
 99     PHI OF RHO IN S-CHAN HEL FRAME
100  24          15.      -180.     40
101  759         10
102  FINISH      1
103  EVA         25
104     INVARIANT MASS OF P PI-
105  100         .04      1.08
106  721         10
107     INVARIANT MASS OF P PI+
108  100         .04      1.08
109  722         10
110     INVARIANT MASS OF PI PI
111  100         .04      .28
112  723         10
113    DELSQ OF RHO
114  100         .02                40
115  731         10
116     COSINE OF RHO IN T-CHAN HEL FRAME
117  40          .05      -1.       40
118  756         10
119     PHI OF RHO IN T-CHAN HEL FRAME
120  24          15.      -180.     40
121  757         10
     ... ... ... ...
     ... ... ... ...

180  FINISH      1
181  *ALL DONE
```

TABLE III

MACRO LIBRARY

```
% MACRO SYMBOL        ; DEFINE DEFAULT VALUES FOR SYMBOLS
% COSL='-1.'          ; LOWER EDGE FOR COS-HIST.
% DCOS=.05            ; CHANNEL WIDTH FOR COS-HIST.
% DDEL=.02            ; CHANNEL WIDTH FOR DELSQ-HIST.
% DELL=''             ; LOWER EDGE FOR DELSQ-HIST.
% DM=.04              ; CHANNEL WIDTH FOR MASS-HIST.
% DPHI=15.            ; CHANNEL WIDTH FOR PHI-HIST.
% FAC=''              ; FACTOR TO SCALE HIST.
% FOLD=''             ; 'FOLD' OR ANYTHING TO FOLD BLOCK7 PLOT.
% NBIT=4              ; NBITS FOR BLOCK7 PLOT.
% NCOS=40             ; CHANNEL NUMBER FOR COS-HIST.
% NDEL=100            ; CHANNEL NUMBER FOR DELSQ HIST.
% NM=100              ; CHANNEL NUMBER FOR MASS HIST.
% NPHI=24             ; CHANNEL NUMBER FOR PHI HIST.
% NPT=''              ; PRINCIPAL TEST NUMBER
% NT=''               ; TEST ASSOCIATED WITH MULTIPLICITY ELEMENT.
% NT2=''              ;
% NT3=''              ;
% NT4=''              ;
% LOG=''              ; 'LOG' OR ANYTHING TO GET LOG HIST.
% PHIL='-180.'        ; LOWER EDGE FOR PHI HIST.
% SGM=''              ; LOCATION OF ERROR (SIGMA).
% SGM2=''             ;
% SGM3=''             ;
% SGM4=''             ;
% WGT=''              ; LOCATION OF WEIGHT
% WGT2=''             ;
% WGT3=''             ;
% WGT4=''             ;
% XL=''               ; LOWER EDGES FOR BLOCK7 PLOT
% YL=''
%; DEFINE BLOCK6 MACROS         ------------------------------------------
% MACRO ONE(TITLE,N,DX,XL,X)
  &&TITLE
&&N        &&DX       &&XL       &&NPT      &&FAC                              &&LOG
&&X        &&WGT      &&NT       &&SGM
% MACRO MASS(SYS,ML,X)
% CALL ONE(' INVARIANT MASS OF &SYS',NM,DM,ML,X)
% MACRO COS(SYS,X)
% CALL ONE(' COSINE OF &SYS',NCOS,DCOS,COSL,X)
% MACRO PHI(SYS,X)
% CALL ONE (' PHI OF &SYS',NPHI,DPHI,PHIL,X)
% MACRO DELSQ(SYS,X)
% CALL ONE ('DELSQ OF &SYS',NDEL,DDEL,DELL,X)
%; BLOCK7 MACROS                ------------------------------------------
% MACRO TWO(SYSX,SYSY,X,Y)
  &&SYSX       VS.     &&SYSY
&&NPT                 &&NBIT     &&FOLD
&&NX       &&NY       &&DX       &&DY       &&XL       &&YL
&&X        &&Y        &&NT
%; DEFINE SELECT MACROS         ------------------------------------------
% MACRO TEST(O,N,M,A,B,C)
TEST       &&N                                         &&C
           &&M        &&O        &&A        &&B
% MACRO BET(N,M,A,B,C)
% CALL TEST('BET',N,M,A,B,C)
% MACRO BIG(N,M,A,C)
% CALL TEST('BIG',N,M,A,'',C)
% MACRO EQU(N,M,A,C)
% CALL TEST('EQU',N,M,A,'',C)
% MACRO EQU2(N,M,T1,T2,C)
% CALL EQU(N,M,T1,C)
           &&M        EQU        &&T2
% MACRO TRUE(N,M,C)
% CALL TEST('TRUE',N,M,'','',C)
```

Table III (continued)

```
% MACRO MT2(O,N,T1,T2,C)
% CALL TRUE(N,T1,C)
&&O        &&T2       TRUE
% MACRO MT3(O,N,T1,T2,T3,C)
% CALL MT2(O,N,T1,T2,C)
&&O        &&T3       TRUE
% MACRO MT4(O,N,T1,T2,T3,T4,C)
% CALL  MT3(O,N,T1,T2,13,C)
&&O        &&T4       TRUE
% MACRO SET2(T1,T2)
% NT=T1
% NT2=T2
% MACRO SET4(T1,T2,T3,T4)
% CALL SET2(T1,T2)
% NT3=T3
% NT4=T4
% MACRO NULLT
% NPT=''
% CALL SET4(,,,)
%; INCIDENT ENERGY INTEVALS FOR PHOTO-PROD. EXP   ---------------------
% MACRO E(N,A,B)
% CALL BET(N,E1,A,B,'')
% MACRO EINT
% CALL E(22,4.5,5.5)
% CALL E(23,5.5,7.)
% CALL E(24,7.,9.)
% CALL E(25,9.,12.)
% CALL E(26,12.,30.)
% CALL E(29,5.5,30.)
%; TWINKLE LITTLE STAR COMMON TO MOST SUMX JOBS   ---------------------
% MACRO TWINKLE(A,B,C)          ; PROLOGUE
*NEW PASS &&A
 &C
*DISCARD  &&B
*TAPE
 10
*SELECT
% MACRO LITTLE(A)               ; SELECT BY WEIGHT AND PROBABILITY
% CALL BIG(14,10,0,'WEIGHT')
% CALL BIG(12,7,A,'PROB')
% CALL MT2('AND',1,14,12)
% CALL EQU(16,'-14','-1','RJCT BY ION')
% CALL EQU(18,'-14',1,'PROTON IDENT BY ION')
% MACRO STAR                    ; SET UP 4-VECTOR BANK
% E1=701
*CHARM
SETUP               8           &&E1
*SELECT
% CALL EINT                     ; INCIDENT ENERGY INTEVALS
%; SYMBOLS AND CHARMS FOR 1 + 2 ... 3 + 4 + 5     ---------------------
% MACRO MCHM(IX)
M&&IX               2           &&M&&IX    &&N       &&IX                &&E1
% MACRO CCHM(IX)
C&&IX               3           &&C&&IX    &&N       &&IX      &&I       &&E1
% MACRO ACHM(IX,IY)
A&&IX               4           &&A&&IX    &&N       &&IX      &&IY      &&E1
%  MACRO C33
% M34=721           ; INVARIANT MASSES
% M35=722
% M45=723
% C34=724           ; PRODUCTION COSINES AND DELSQ'S
% D34=725
% C35=727
% D35=728
% C45=730
```

Table III (continued)

```
% D45=731
% A34=734                ; DECAY ANGLES
% A35=744
% A45=754
% CJ45=756               ; J (H) REFERS TO T-CHANNEL (S-CHANNEL) HELICITY FRAMES
% PJ45=757               ; C (P) FOR COS (PHI)
% CH45=758
% PH45=759
% CW45=760
% PW45=761
% N=2                    ; NO OF PARTICLES
*CHARM
% CALL MCHM(34)
% CALL MCHM(35)
% CALL MCHM(45)
COSP-     0         3         &&C34     1         5         1         &&E1
COSP+     0         3         &&C35     1         4         1         &&E1
COS+-     0         3         &&C45     1         3         2         &&E1
% CALL ACHM(45,12)
% CALL ACHM(34,21)
% CALL ACHM(35,21)
%; MACROS TO SELECT CHANNELS  -----------------------------------------
% MACRO H2(N1,N2,N3,I1,I2,D)
% CALL EQU2(N1,'-4',I1,I2,'&D.UNIQUE')
% CALL EQU2(N2,'-4','-&I1','-&I2','&D.AMB')
% CALL MT2('',N3,N1,N2,'&D.ALL')
% MACRO T33
% CALL H2(2,3,4,301,302,'P PI+ PI-')
% CALL H2(6,7,8,303,304,'P K+ K-')
% CALL EQU2(9,'-4',305,'-305','P PBAR P')
```

of these is to define the master test (TEST 1) to eliminate unwanted events before CHARM calls are made to calculate various quantities to be SUMXed. The latter is done by a call (line 8A) to C33 (lines 122-151L). As shown C33 in particular assigns BOUT locations to various names for quantities being calculated.

MCHM (lines 116-117L) shows how a symbol can be concatenated and then substituted. Note here that the variable N is not included in the parameter list. Since it is comparatively slowly varying it is set by assignment (line 141L) rather than explicitly as an argument of the macro. C33 expands into lines 63-72B. For completeness explanations for CHARMs used are given in Appendix B.

Unassigned escape names, i.e., those which are preceded by & but have not yet appeared in assignment statements are left unchanged in the text. This choice rather than automatic null-assignment is made because of possible use in other language such as in FORTRAN IV in which the character & is used to designate statement labels. The purpose of SYMBOL (lines 1-31L) is to assign default values to symbols occurring globally as in the BLOCK6 and 7 macros (lines 33-50L), any of which may be set and reset before use as in lines 13, 22, and 26A.

Macros can, of course, be defined outside the library by the user as in lines 14-28A. The corresponding calls (lines 29-32A) result in SUMX control statements for a set of histograms repeated for different incident energy intervals as shown in lines 77-180B.

It was often asked, "in this scheme how many lines can be produced by a single line of typing?". The answer is obviously, "one-to-all", because all of the lines say, in this example can be lumped into a single macro. The need for doing so may conceivably arise in on-line applications.

It is also said that some of these features are available through additional programming in SUMX. This must be self-evident, because MACROS itself is a piece of program. The idea being propounded is in the use of a <u>general</u> purpose macro processor. In this respect it would have been better, if the "macro part" of the 360 assembler could easily be detached so that it could also be used elsewhere.

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                          MAC01  8SEP69

  1      0                %;
  2      0                %;
  3      0                %;
  4      0                %;
  5      0                %;
  6      0                %;
  7      0                %;
  8      0                %;
  9      0                %;
 10      0                %;
 11      0                %;
 12      0                %;
 13      0                %;
```

Trivial syntax errors are evident from diagnostic messages from MACROS. In the following we describe examples of errors which require some explanation.

(1) Scanning for escape names is done from left to right. A peculiar effect can arise from concatenating names. For example in the use of the *MACRO ACHM defined in lines 120-121 of Table III:*

```
 14      0                %          A12=1234
205      0                %          CALL ACHM(12,34)
206      1      5         A12                4          1234      &&N       12        34        &&E1
208      0                %;
```

which is as expected. However,

```
209      0                %;
210      0                %;
211      0                %          A=''
212      0                %          CALL ACHM(12,34)
213      1      5         A12                4            12      &&N       12        34        &&E1
215      0                %;
216      0                %;
```

(2) When a substitution for an escape name is done, the line of text is rescanned (because the value itself may be an escape name) until all possible substitutions are carried out. This can cause a loop as in the following example:

```
217      0                %;
218      0                %;
219      0                %;
220      0                %;
221      0                %;
222                       %          MACRO M1(A)
223      E                           &A  STAR
224      E                %MEND (GENERATED)
225                       %          MACRO M2(A)
226      E                %          CALL M1('&A LITTLE')
227      E                %          MEND
228      0                %;
229      0                %          CALL M2('TWINKLE')
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1              ,MODEL STMT=   223,LAST TOKEN='A'
          SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
```

2105A16

```
                                                                                              PAGE     2
STMT  LEVEL REPLC        SOURCE STATEMENT                                              MACOI   8SEP69
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV= 8     SIG PART OF STMT/REPL VALUE LOST IN PACKED REPL
     ***ERROR IN STMT   231,LEVEL= 2,MACRO=M1                  ,MODEL STMT=   223,LAST TOKEN='A'
         SEV=12     TOO MANY REPLACEMENTS
231     2    31                &A LITTLE LITTLE LITTLE LITTLE LITTLE LITTLE LITTLE LITTLE LITTLE LITT
234     0             %;
235     0             %;
236     0             %;
237     0             %;
238     0             %;
```

which also serves to indicate that the maximum number of replacements per line is 32.

The solution is, of course, to redefine the MACRO as

```
239                   %        MACRO M3(B)
240     E             %        CALL M1('&B LITTLE')
241     E             %        MEND
242     0             %;
243     0             %        CALL M3('TWINKLE')
245     2     2                TWINKLE LITTLE  STAR
248     0             %;
249     0             %;
250     0             %;
251     0             %;
```

(3) *Processor statements are not subject to replacements as in*

```
252                   %        MACRO M(I,C)
253     E             %        CALL M&I (C)
254     E             %        MEND
255     0             %;
256     0             %        CALL M(3,'TWINKLE')
     ***ERROR IN STMT   266,LEVEL=10,MACRO=M                   ,MODEL STMT=   253,LAST TOKEN='M'
         SEV=16     MACRO DEPTH EXCEEDS MAXIMUM

CONDITION ERR  OCCURRED IN STATEMENT 00322 AT OFFSET +00266 FROM ENTRY POINT MACROCALL        2105A17
```

(4) Because of the lack of MACRO-time arithmetic and conditional GOTO statement, it is obvious that a MACRO cannot be defined recursively. This is what happens:

```
STMT  LEVEL REPLC         SOURCE STATEMENT                                    MAC01  8SEP69

 CALLED, IN STATEMENT 00789, FROM PROCEDURE WITH ENTRY POINT MACROS

267     0              %;
268     0              %;
269     0              %;
270     0              %;
271     0              %;
272     0              %;
273                    %        MACRO R(A,B,C)
274     E                       I AM AT LEVEL &A
275     E              %        CALL R(B,C,,)
276     E              %        MEND
277     0              %;
278     0              %        CALL R(1,2,3)
279     1     1                 I AM AT LEVEL 1
281     2     1                 I AM AT LEVEL 2
283     3     1                 I AM AT LEVEL 3
285     4     1                 I AM AT LEVEL
287     5     1                 I AM AT LEVEL
289     6     1                 I AM AT LEVEL
291     7     1                 I AM AT LEVEL
293     8     1                 I AM AT LEVEL
295     9     1                 I AM AT LEVEL
297    10     1                 I AM AT LEVEL
      ***ERROR IN STMT   298,LEVEL=10,MACRO=R                  ,MODEL STMT=   275,LAST TOKEN='R'
           SEV=16    MACRO DEPTH EXCEEDS MAXIMUM

CONDITION ERR  OCCURRED IN STATEMENT 00322 AT OFFSET +00266 FROM ENTRY POINT MACROCALL

 CALLED, IN STATEMENT 00789, FROM PROCEDURE WITH ENTRY POINT MACROS

299     0              %;
300     0              %;
301     0              %;
302     0              %;
303     0              %;

              ---M A C R O S PROCESSING COMPLETED,HIGHEST SEVERITY=16                 2105A18
```

which also serves to show that the maximum level of nested MACRO call is 10.

APPENDIX A

The following is reproduced from Ref. 2.

```
                    III. MACROS Language Description                          PAGE   22
      LEVEL REPLC   SOURCE STATEMENT                                   MAC01  8SEP69

        0       %;       This section will describe the elements of the MACROS
        0       %;  language in detail.  The examples of preceding sections were
        0       %;  of an introductory nature, while this section is designed for
        0       %;  reference purposes.
        0       %;       The syntax notation uses names of items enclosed in < >
        0       %;  symbols to denote syntactic entities.  Definitions of them are
        0       %;  either given in words or in terms of other entities.  This is
        0       %;  indicated by an <item> followed by ::= and definitions.  The
        0       %;  symbol | is used to indicate alternates.
        0       %;
        0       %;  I.  ELEMENTS
        0       %;
        0       %;  <identifier> -- 1-16 alphanumeric characters, the first of
        0       %;       which must be alphabetic.  Letters are A-Z,$,#,@; digits
        0       %;       are 0-9.  Identifiers are used to denote processor
        0       %;       variables and macros.
        0       %;
        0       %;  <integer> -- 1-9 digits.  Its usage in MACROS is identical to
        0       %;       strings, since this version has no macro-time arithmetic.
        0       %;  <string> -- there are two forms:
        0       %;       (1) Text string-- 0-80 characters enclosed in quotes.
        0       %;           A quote within a string is represented by 2 consec-
        0       %;           utive quotes.  The string of no length,or null string
        0       %;           is represented by ''.
        0       %;       (2) Fixed point number--string of digits preceded by, or
        0       %;           followed by, or containing a decimal point.  This
        0       %;           construct is included for convenience in writing
        0       %;           MACROS statements; but it is not a number and does
        0       %;           not possess a numeric value.  It should be thought
        0       %;           of as if it were enclosed in quotes.
        0       %;
        0       %;  <constant> ::= <integer> | <string>
        0       %;
        0       %;  <formal parameter> ::= <identifier>
        0       %;       The formal parameter is declared by appearance in a
        0       %;       MACRO statement.  Its value is defined when the macro
        0       %;       is expanded--rules are given below.
        0       %;
        0       %;  <simple variable> ::= <identifier>
        0       %;       Simple variables are declared and assigned values
        0       %;       by appearance on the left side of an assignment statement.
        0       %;       They assume the type of the right side of the statement--
        0       %;       in this version of MACROS it can always be considered
        0       %;       string type since no macro-time arithmetic is permitted.
        0       %;
        0       %;  <processor variable> ::= <simple variable> | <formal parameter>
        0       %;
        0       %;  <processor expression> ::= <constant> | <processor variable>
        0       %;       The processor expression is used on the right side of
        0       %;       assignment statements or in actual parameters of CALL
        0       %;       statements.  Its value is either the string value of
        0       %;       the constant or the value currently associated with
        0       %;       the processor variable.  If the variable has no value
        0       %;       associated with it, then the expression is erroneous,
```

```
LEVEL REPLC           SOURCE STATEMENT                                        MAC01  8SEP69

  0           %;          and its value is taken to be the null string.
  0           %;
  0           %;     Examples:
  0           %;          <identifier> --  ABCD   $$$   OTHER1
  0           %;          <integer> --  0  123  123456789
  0           %;          <string> --  'TEXT STRING'  'PEOPLE''S PARK'  ''
  0           %;                      .123  12.3  456.   (Fixed point number)
  0           %;          <processor expression> --   ABCD  123   .9   'STRING'
  0           %;
  0           %;
  0           %;     II.  PROCESSOR STATEMENTS
  0           %;
  0           %;     General:
  0           %;          Processor statements always have the character '%' in
  0           %;     column 1.  The body of the statement must be in columns 2-72.
  0           %;     No continuation statements are allowed.  With the exception of
  0           %;     assignment and null statements, they begin with a reserved word
  0           %;     followed by an operand field.  Comments may follow the
  0           %;     logical end of the statement when separated by a semi-colon
  0           %;     (;).  Blanks may be used freely between elements and keywords,
  0           %;     they are only required to separate alphanumeric items.
  0           %;          Processor statements are not subject to replacement (section
  0           %;     IV) or may they be generated by replacement.  Thus if a '%' is
  0           %;     generated by replacement in column 1, MACROS will not recognize
  0           %;     the statement as a processor statement.
  0           %;          Processor statements are always printed at top level and
  0           %;     while a macro is being edited.  They are never printed at lower
  0           %;     levels (i.e. within a macro expansion).
  0           %;
  0           %;    Null or comment statement--
  0           %;     Form:      % ;  comments
  0           %;          The null statement is ignored by MACROS.  When in a macro,
  0           %;     it is not encoded, so no space is required for it in the
  0           %;     edited macro.
  0           %;     Example:
  0           %;          %;The report you are reading is mostly null statements
  0           %;
  0           %;    MACRO statement
  0           %;     Form:      %  MACRO  <macroname> (<formal parameter list>)
  0           %;          or    %  MACRO  <macroname>
  0           %;                <macroname> ::= <identifier>
  0           %;                <formal parameter list> ::=  <formal parameter> |
  0           %;                     <formal parameter list> , <formal parameter>
  0           %;          The MACRO statement heads a macro definition.  The macro-
  0           %;     name is used in CALL statements to reference the macro.  It may
  0           %;     be the same as a processor variable identifier.  No two macros
  0           %;     may have the same name.
  0           %;          The formal parameter list, if supplied declares the
  0           %;     identifiers as formal parameters.  A maximum of ten may be given.
  0           %;     Details of their interpretation are in the next section.
  0           %;          Macros may not be nested.  The macro definition ends at
  0           %;     a MEND statement, the next MACRO statement, or end-of-file.
  0           %;     Examples:
  0           %;          %  MACRO  FIBONACCI(I)
  0           %;          %  MACRO  $DECLARATIONS
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                          MAC01  8SEP69

        0           %;           %  MACRO  FUNCTION   ( TYP,ARG )
        0           %;
        0           %;  MEND statement
        0           %;   Form:      %  MEND
        0           %;        The MEND statement terminates a macro definition that
        0           %;   has not been otherwise ended.
        0           %;
        0           %;   CALL Statement
        0           %;   Form:      %  CALL  <macroname> (<actual parameter list>)
        0           %;        or    %  CALL  <macroname>
        0           %;              <actual parameter list> ::=  <actual parameter>  |
        0           %;                   <actual parameter list> , <actual parameter>
        0           %;              <actual parameter> ::= <processor expression>
        0           %;        The CALL statement invokes the macro <macroname>,
        0           %;   which must have been previously defined or be available on file
        0           %;   SYSLIB.
        0           %;        The actual parameter list if present is used to assign
        0           %;   values to the formal parameters.  Details are given in the next
        0           %;   section.
        0           %;   Examples:
        0           %;        %   CALL   FIBONNACI ( 1 )
        0           %;        %   CALL   $DECLARATIONS
        0           %;        %   CALL   FUNCTION ( 'SIN' , 'X+Y' )
        0           %;
        0           %;  TITLE statement
        0           %;   Form:      %  TITLE    <string>
        0           %;          or  %   TITLE
        0           %;        The TITLE statement sets the page title to <string> if
        0           %;   present and causes a page eject.  Omitted operand leaves the
        0           %;   title unchanged and causes a page eject only.  The statement
        0           %;   is not processed within a macro definition--only when it is
        0           %;   expanded.  The TITLE statement is never printed.
        0           %;   Example:
        0           %;        %   TITLE  'III. MACROS Language Description'
        0           %;        The above was used to get the current page title.
        0           %;
        0           %;  Assignment statement
        0           %;   Form:      %  <leftside> =  <processor expression>
        0           %;              <leftside> ::=  <processor variable>
        0           %;        The assignment statement is used to set the value of the
        0           %;   leftside to the value of the processor expression.  If the
        0           %;   leftside is a processor variable, then the statement also
        0           %;   declares the identifier as such if it has not been used
        0           %;   before.  If it is a formal parameter, then the assignment
        0           %;   is only retained during this expansion of the macro.  The
        0           %;   actual parameter, if it was a processor variable, will not
        0           %;   be changed by the assignment.
        0           %;
        0           %;
        0           %;   III. MACRO EXPANSIONS
        0           %;
        0           %;        When a macro is expanded using the CALL statement, the
        0           %;   following actions take place:
        0           %;   (1)  If any formal parameters were declared, any values currently
        0           %;        assigned to the formal parameter identifiers are saved on a
```

```
0   %;       pushdown stack.
0   %;  (2)  The values of the actual parameters supplied in the
0   %;       CALL statement are assigned to the formal parameter
0   %;       identifiers.  If any or all of the actual parameters are
0   %;       omitted, then the corresponding formal parameters are
0   %;       set to the null string.  Excess actual parameters are
0   %;       ignored.  Thus a call statement of the form
0   %;            % CALL FUNCTION(,3)
0   %;       for the macro FUNCTION above, will set TYP='' and ARG='3'.
0   %;  (3)  MACROS begins fetching input from the body of the macro
0   %;       definition.  The processor variables that are currently
0   %;       active will be used during processing--this includes
0   %;       any variables set by macros that called this one.
0   %;  (4)  When the end of the macro definition is reached, the
0   %;       values previously associated with the formal parameter
0   %;       identifiers are restored with the values saved on the
0   %;       pushdown stack, if any.  MACROS then resumes fetching
0   %;       statements in the environment of the invoking CALL
0   %;       statement; but changes made to variables global to the
0   %;       called macro are retained.
0   %;  The maximum depth of macro calls is 10 levels.
0   %;
0   %;
0   %;  IV.  REPLACEMENT IN BASE LANGUAGE STATEMENTS
0   %;
0   %;       Every base language statement (i.e. no '%' in col 1)
0   %;  outside of a macro definition is scanned by MACROS for the
0   %;  escape character '&', which signals a potential replacement.
0   %;  If the '&' is followed by an identifier or by another '&' and
0   %;  an identifier, and if the identifier is that of a processor
0   %;  variable that is currently active, then a replacement is per-
0   %;  formed.  The single ampersand is used to denote "packed"
0   %;  replacement, for free format applications; and the double
0   %;  ampersand for "non-packed" replacement, for use where column
0   %;  positions are critical.
0   %;       The term "escape name" is used to refer to the escape
0   %;  character(s) and the immediately following identifier.  For one
0   %;  case of packed replacement, a period delimiter is included in
0   %;  the escape name.
0   %;       Scanning for an escape character proceeds from left to
0   %;  right.  The scanning field is the entire Fortran statement,
0   %;  including continuation lines, if the FORT=1 option is used;
0   %;  else it is the columns between the begin and end columns (inclu-
0   %;  sive) of the 80 character record.  The default scanning columns
0   %;  are 1 thru 80; they may be changed by the user if needed.
0   %;       When an escape name if found, and replacement is done (the
0   %;  identifier has a value), then the scan is restarted from the begin
0   %;  column.  If the identifier does not have a value, then the scan
0   %;  proceeds to the right.  Rescanning continues until no
0   %;  more escape characters remain, or no more replacements can be
0   %;  done.  The rescanning mechanism allows construction of escape
0   %;  names by replacement.  The maximum number of replacements allowed
0   %;  in a base language statement is 30.
0   %;       Specific rules for the replacement modes are as follows:
0   %;
```

(1) Packed replacement--This is indicated by a single '&' followed by an identifier, delimited by any special character. If it is delimited by '.', then the period will also be removed from the text--this allows concatenation of a replacement value with a letter or digit. Replacement is done by removing the escape character and the identifier (and period, if necessary) from the text, and inserting the value associated with the identifier in its place. The remainder of the statement to the right of the escape name , extending to the end of the scanning field, is shifted right or left as needed to accommodate the replacement value. Blanks will be added at the end of the statement if the value is shorter than the escape name, or truncation will be preformed if the value is longer. In the latter case, loss of non-blank characters will be noted by MACROS.

(2) Non-packed replacement--This is denoted by a double escape character '&&' followed by an identifier. The identifier is delimited by any special character (no special rule for period applies). The replacement of the escape name is done without any shifting of the part of the statement to the right of the escape name. If the replacement value is shorter than the escape name, then blanks are added to it. If the value is longer, then blank positions following the escape name are used to hold the trailing non-blank characters of the value. If there is still not enough room, even using the blank positions, and discounting trailing blanks of the value, then MACROS notes the truncation of the value.

```
0    %;  MACROS is a load module named MACRO01 which can be used
0    %;  from the library PUB.JEA.LOADMODS.  It uses the following
0    %;  ddnames:
0    %;
0    %;  SYSPRINT -- output listing. DCB information is supplied by the
0    %;       program--RECFM=VB,LRECL=125,BLKSIZE=3425.  The PGEN option
0    %;       may be set to suppress the output listing in whole or part.
0    %;  SYSIN -- primary input file. May be any sequential dataset
0    %;       or partitioned dataset member (or concatenation) with
0    %;       logical record length 80 bytes.
0    %;  SYSOUT -- MACROS processed output.  DCBs must be supplied by
0    %;       user such that LRECL=80 is used.  Generally, this file
0    %;       is associated with a temporary dataset that is used later
0    %;       in the job.  An option may be given to MACROS to suppress
0    %;       writing on SYSOUT.
0    %;  SYSLIB -- secondary input file for macro definitions.  This
0    %;       is used to make a library of macro definitions available
0    %;       to MACROS.  If not required, then the DD statement can be
0    %;       omitted.  If used, the DD statement must define a sequential
0    %;       dataset or partitioned dataset member.  Concatenations of
0    %;       these two items are acceptable if the DCBs are the same.
0    %;       The logical record length must be 80 bytes.
0    %;
0    %;       Options are passed to MACROS using the PARM field on the
0    %;  EXEC statement.  The format is a list of keywords followed by
0    %;  an equal sign and an integer, separated by commas.  The options
0    %;  and their default values are:
0    %;
0    %;  BEGC (1)        Begin column for scanning of base language state-
0    %;                  ments.  This is ignored if the FORT=1 option is
0    %;                  used.
0    %;  ENDC (80)       End column for scanning of base lanugage state-
0    %;                  ments.  Also ignored if FORT=1.
0    %;  MOUT (1)        If not zero, then the MACROS output is written
0    %;                  on SYSOUT.
0    %;  PGEN (2)        If zero, then no output listing is produced.
0    %;                  If one, then top level statements only are listed.
0    %;                  If two, then base language statements from macro
0    %;                  expansions are listed in addition to top level
0    %;                  statements.
0    %;  PLIB (0)        If not zero, then the macro definitions edited
0    %;                  from SYSLIB are listed when it is opened.
0    %;  FORT (0)        If not zero, then the FORTRAN statement conven-
0    %;                  tions are used--columns 1-72 of the first card of
0    %;                  a statement and columns 7-72 of any continuation
0    %;                  cards are treated as a single base language
0    %;                  statement for replacement scanning.  If there are
0    %;                  no Hollerith literals in quotes in the statement,
0    %;                  then trailing blanks at the end of each line are
0    %;                  removed.  There is a limit of 270 characters that
0    %;                  may be retained, discounting trailing blanks.
0    %;                  Thus, in the worst case, 3 continuation cards are
0    %;                  allowed.
0    %;  PRBS (0)        If not zero, then 'PRint Before Substitution',
```

```
0   %;                  which causes base language statements containing
0   %;                  escape characters to be listed before replacement
0   %;                  is done.  Normally, they are listed after replace-
0   %;                  ment only.
0   %;  SIZS (10000)    The amount of space (in bytes) to be allocated to
0   %;                  storage of the values of variables.  Should one
0   %;                  find that he has run out (error message), he can
0   %;                  rerun with a larger value (up to 32767).  Space
0   %;                  could run out if a large number of variables are
0   %;                  all active at the same time.
0   %;
0   %;       When MACROS is run, the first page of the listing gives the
0   %;  options used, and a number 'SIZW', which is the amount of free
0   %;  space in bytes that MACROS thinks it has for storage of strings,
0   %;  macro definitions and control information.  MACROS will use all
0   %;  the free space in the region in which it is run.  Minimum region
0   %;  size is about 100k bytes.
0   %;
0   %;       Errors in MACROS input are flagged with a text message
0   %;  indicating the type of error,disposition, and location.
0   %;  Severities are associated with the errors ranging from 4-16.
0   %;  The highest severity is passed back to OS as a return code
0   %;  that can be tested in JCL to suppress subsequent job steps.
```

APPENDIX B

CHARMS

An event in our SUMX is represented by a FORTRAN array P(4, N) consisting of four-vectors, one for each participant of an N-particle reaction,

$$1 + 2 \rightarrow 3 + 4 + \ldots$$

where numbers correspond to the second index of the array P. As discussed in the text the purpose of the CHARMs to be described here is to calculate from P various quantities of interest, which are to be specified at the SUMX run time.

As described in the SUMX manual[1] a CHARM control-card allows five parameters, L1, L2, . . . , L5, by means of which user can communicate with the requested CHARM subroutine. In the descriptions below "pointer" means BOUT location, a group of particles is denoted by ij. . . , the total number in this group by N. The CHARM parameter L5 is always the base pointer for the array P.

(1) CHARM2

Invariant mass and four-momentum transfer for a group of particles.

L1 : pointer for answer

L2 : N

L3 : ij. . .

L4 : 0 to get $M = \mathrm{SQRT}\left[(P_i + P_j + \ldots)^2\right]$,

-n to get $\Delta^2 = -(P_i + P_j + \ldots\ -P_n)^2$.

(2) CHARM3

Four-momentum transfer (Δ^2), production cosine (COS) and momentum (Q) for a group of particles in the overall center-of-mass.

L1 : base pointer for answer vector A

A(1)=COS ,

A(2)=$\Delta^2 - \Delta^2_{min}$ (kinematic minimum),

A(3)=Q (calculated only if L1 is negative) .

L2 : N

L3 : ij. . .

L4 : 1 or 2 to specify with respect to beam or target.

(3) CHARM4

Two or three body decay angles.

L1 : base pointer to answer vector A

L2 : N (2 or 3)

L3 : ij...

L4 : ab (12 or 21 depending on where 1 or 2 is to be incident)

The calculated decay angles have the following meaning. Let

$$R = P_i + P_j + \dots$$

and T be such that

$$a + b = R + T \quad .$$

Let $\hat{y}$ be the normal to the production plane,

$\hat{y}$ // $\vec{T} \times \vec{a}$ in the R-rest frame, or

// $\vec{a} \times \vec{R}$ in the laboratory frame .

Then in the rest frame of R define the t-channel helicity (Gottfried-Jackson) axes as

$$\hat{z} = \hat{a}$$

and

$$\hat{x} = \hat{y} \times \hat{z}.$$

The s-channel helicity axes are related to the above by a rotation about the common y-axis. Let

$$\hat{z}_H = \hat{R} \text{ in the overall center-of-mass,}$$

or

$$= -\hat{T} \text{ in the R-rest frame ,}$$

and

$$\hat{y}_H = \hat{y} \quad ,$$

$$\hat{x}_H = \hat{y}_H \times \hat{z}_H \quad .$$

Then using i as the analyzer CHARM4 calculates the following

$$A(1) = \hat{z}_H \cdot \hat{z}$$

$$A(2) = \tan^{-1} (\hat{z}_H \cdot \hat{x}/\hat{z}_H \cdot \hat{y}) \; ,$$

$$A(3) = \hat{i} \cdot \hat{z} \quad ,$$

$$A(4) = \tan^{-1} (\hat{i} \cdot \hat{x}/\hat{i} \cdot \hat{y}) \quad ,$$

$$A(5) = \hat{i} \cdot \hat{z}_H \quad ,$$

$$A(6) = \tan^{-1} (\hat{i} \cdot \hat{x}_H/\hat{i} \cdot \hat{y}_H),$$

where the angles are in degrees.

It appears in this scheme that because SUMX treats L3, L4 as integers particle index greater than 9 cannot be accommodated. One solution, as long as they need not be addressed at once, is to move base pointer L5 within P or to rearrange members of P before use. Far better solution is to "ask" SUMX to regard these parameters as character strings, so that a number system of arbitrary base can be employed.

APPENDIX C

JCL EXAMPLES

We give examples of JCL statements necessary for runs on the System 360/91 as implemented at SLAC and at MPI. Catalogued procedures used are current ones in May 1972.

(1) At SLAC

```
//  JOB card
//JOBLIB DD DSN=WYL.ED.PUB.LIB9,DISP=(SHR,PASS)
//MACRO EXEC PGM=MACRO01
//SYSPRINT DD SYSOUT=A
//SYSOUT DD DSN=&MOUT,DISP=(NEW,PASS),UNIT=SYSDA,
//  SPACE=(TRK,(200,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSLIB DD DSN=WYL.ED.JAP.SRC9(SMCR1),DISP=SHR
//         DD  user macro library
//SYSIN DD *
```

input text

```
//SUMX EXEC FORTHCLG,PARM.FORT='OPT=2'
//FORT.SYSIN DD*
```

user FORTRAN source, if any

```
//LKED.SYSLIB DD DSN=WYL.ED.PUB.LIB9,DISP=SHR
//              DD DSN=SYS1.FORTLIB,DISP=SHR
//              DD DSN=SYS3.FORTLIB,DISP=SHR
//              DD DSN=SYS4.FORTLIB,DISP=SHR
//LKED.SYSIN DD *
    INCLUDE SYSLIB(SUMX)
    ENTRY MAIN
//GO.FT10F001 DD  user  DST  description
//GO.SYSIN DD DSN=&MOUT,DISP=(OLD,DELETE)
```

(2) <u>At MPI</u>

```
//   JOB card
//D EXEC PGM=MACRO01,REGION=300K
//STEPLIB DD DSN=LOAD.JHP,DISP=SHR
//SYSLIB DD DSN=SOR.JHP(MACLIB),DISP=SHR
//          DD   user macro library
//SYSPRINT DD SYSOUT=A
//SYSOUT DD DSN=MOUT,DISP=(NEW,PASS),UNIT=DISK,
//   SPACE=(TRK,(200,10)),DCB=SOR.JHP
//SYSIN DD *
```

input text

```
//S EXEC SUMX
//C.SYSIN DD *
```

user FORTRAN source, if any

```
//L.SYSLIB DD DSN=LOAD.JHP,DISP=SHR
             DD DSN=SYS1.FORTLIB,DISP=SHR
//G.FT10F001 DD user DST description
//G.SYSIN DD DSN=MOUT,DISP=(OLD,DELETE)
```

REFERENCES

1. Although the technique is evidently general, we discuss a particular application to the CERN version of the SUMX described in J. Zoll, CERN Track Chamber Program Library Manual (1970).
2. John Ahern, MACROS-Statement Oriented Macro Processor, SLAC Computation Group User Note 29 (1969).
3. P. J. Borwn, A Survey of Macro Processors, Annual Review in Automatic Programming, Vol. 6, Part 2 (Pergamon Press, New York, 1969).